Midterm I Exam, Fall 2018 **Saturday October 9th, 2018** **Exam time: 06:00-7:30 P.M.**

Student's name:... **ID:** .... **Section:** ....

SOLUTION

Problem 1 (8 points)

(a) Give the following functions a number in order of increasing asymptotic growth rate. If two functions have the same asymptotic growth rate, give them the same number.

Function	Rank
$8^{\lg n} + 2n$	3
$5 \lg n^8$	1
$n^3 + 5n^2 - 100$	3
$\frac{n^2}{\lg n}$	2
$2 \lg n + \lg(\lg n^2)$	1
3^{2n}	4

(b) Using the definition of θ, find $g(n)$, C_1, C_2, and n_0 in the following:

$$4n^6 - 2n^2 + n \in \theta(g(n))$$

Sol: $C_1=7$, $C_2 = 3$, $n_0=1$

Problem 2 (5 points)

What is the time complexity of the following algorithm? Prove your answer.

```
S(A[], key, imin, imax)
{
  if (imax < imin)
        return KEY_NOT_FOUND;
  else
    {
      imid = midpoint(imin, imax);

      if (A[imid] > key)
        return S(A, key, imin, imid-1);
      else if (A[imid] < key)
        return S(A, key, imid+1, imax);
      else
        return imid;
    }
}
```

Problem 3 (7 points)

Consider the following recurrence relation:

$$\mathbf{T(n) = 3T(\frac{n}{3}) + 2n}.$$

Solve this recurrence relation using recursive substitutions and find its asymptotic performance.

Sol: T(n) = 3 T(n/3) + 2n

= 3 (3 T (n/9) +2 n/3) + 2n = 9 T(n/9) + 2n + 2n

= 9 (3 T(n/27) + 2 n/9)) + 2n + 2n

= 27 T(n/27)+ 2n + 2n +2n

…….

$=2^k \quad T\left(\frac{n}{2^k}\right) + 2 \quad k\,n$

We stop when $\frac{n}{2^k} = 1$ → T(n) = n T(1) +n log(n)

Then prove that T(n) = Θ(n lg n)

Problem 3 (4.5 points)

Solve the following recurrences using the Master theorem by giving tight θ-notation bounds. Justify your answers.

(a) **T(n) =4T(n/4) + 5 $\boldsymbol{log^2}$ ($\boldsymbol{n}$)**

Sol: Case 1 → T(n) = θ(n)

(b) **T(n) =9T(n/3) + 3$\boldsymbol{n^2}$**

Sol: Case 2 → T(n) = θ(n^2 $log\, n$)

(c) **T(n) = 7T(n/2) + $\boldsymbol{n^3\, log\, n}$**

Sol: Case 3 → T(n) = θ(n^3 log n) NB: verify constraint

Problem 4 (4.5 points)

For each algorithm listed below, give a recurrence that describes its worst-case running time, and give its worst-case running time using O-notation.

You need **not** justify your answers.

(a) Merge sort
Sol: T(n)= 2T(n/2) + n
O(n log n)

(b) Quicksort algorithm
Sol: T(n)= T(n-1) + n
O(n^2)

(c) Binary Search
Sol: T(n)= T(n/2) + 1
O(log n)

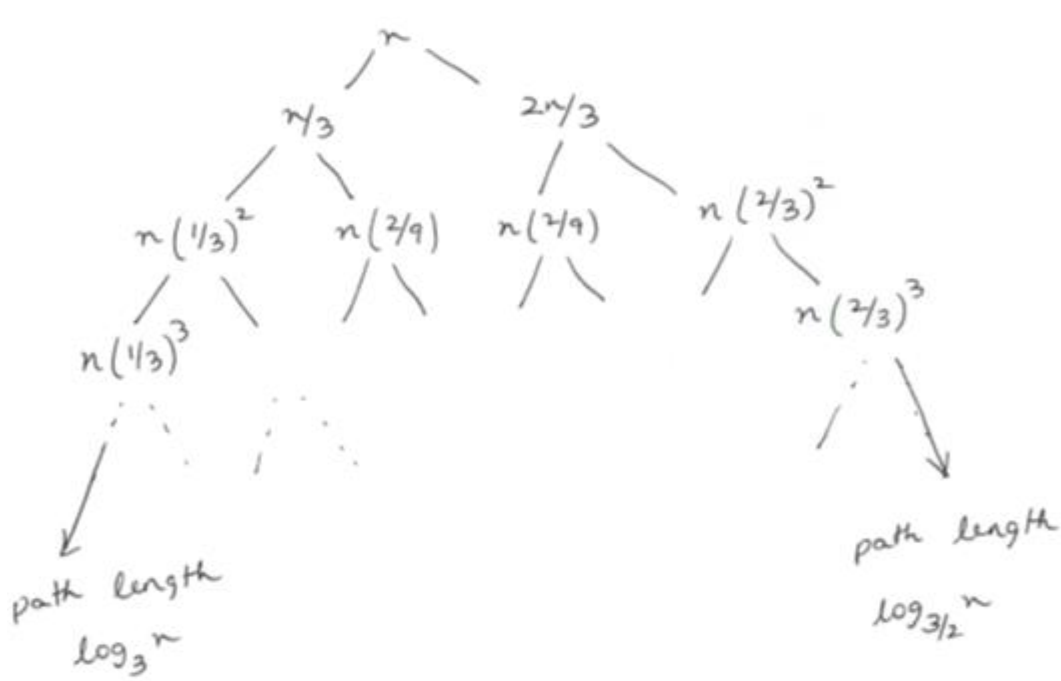
Problem 5 (5 points)

Consider a variation of MergeSort which divides the list of elements into two lists of size 1/3 and 2/3, recursively at each step, instead of dividing it into halves. The Merge procedure does not change.

(a) Give a recurrence relation for this algorithm

Sol: T(n)= T(n/3) + T(2n/3) +n

(b) Draw a recursion tree for the algorithm



(c) Using the recursion tree, explain how you can deduce the worst case upper bound.

Sol: The total work at each level is ≤ n. The longest root-to-leaf path is n – 2n/3 – 4n/9 ---, which is of length $log_{3/2}\ n$ since each level has ≤ n work, total work is ≤ n $log_{3/2}\ n$, which is O(n log n)

Problem 6 (6 points)

Consider the following problem:

There are **n** parking spots numbered from 1 to ***n*** and you are told that there are ***k*** cars in the first ***k*** spots (one car in each spot), and all other spots are empty. How to find the value of ***k***?

a- Suggest **TWO** algorithm design techniques and give a high-level description of the **TWO** algorithms to solve the problem (find ***k***)?
b- Analyze the complexities of your **TWO** algorithms?

Sol:

The first approach is the brute force approach. The basic idea consists of scanning all spots and finding the first empty one:

```
Algorithm findK(A[1..n])
{
    for i← 1..n
        If (A[i] empty) return i
    end
 }
O(n).
```

The second approach is the divide and conquer approach.

```
Algorithm findK(A[l..r])
{
If(l==r) return l;
  m = (l+r)/2
            If (A[m] empty) findK(A[l..m-1])
            else findK(A[m .. r])
            end
 }
O(log n).
```